ways is to use the *auto-increment* mode. The instruction:

```
        LDA     TABLE,X+
```

will cause the value in X to be automatically incremented after it has been used. If we have a table of 16-bit values then we use:

```
        LDA     TABLE,X++
```

which causes the X register to be incremented by two. Our original program loop is now considerably streamlined:

```
LOOP    LDA     TABLE,X+
        INC     COUNT
        LDB     COUNT
        CMPB    #64
        BLT     LOOP
```

Another useful alternative to the original method we outlined is to step through the table of values in the reverse order, perhaps using the auto-decrement mode. This has the advantage that the final value in the X register is zero, and as auto-decrement of an index register automatically sets the flags in the condition code register, we can test directly for the end of the loop without having to use a CMP instruction. The same effect can be obtained by loading the index register with a negative value and incrementing this until zero is reached. Every time the auto-increment instruction is obeyed, it sets the condition code register flags to show the results of the increment. If a zero result occurs, for example, the zero flag is set; if a carry occurs, then the carry flag is set, and so on.

We should remember, however, our general rule that it is always best to make tests on the accumulators only. Also, since most programs are likely to have some processing between the increment/decrement instruction and the test instruction it is unlikely that the condition code register will remain unchanged between the action and the test.

If we decide not to step backwards through the table, it is still a good idea to make the count go backwards so that we can end the loop at zero. A point to watch with the auto-decrement instruction mode is that the decrementing is performed *before* the address calculation, whereas in auto-increment the register is incremented *after* the address has been calculated. Thus, if X contains 7 and TABLE begins at $1000, then the instruction LDA TABLE,X+ will load the accumulator from address $1007 and then increment X from 7 to 8. LDA TABLE,-X (note the minus sign comes before the register name), on the other hand, will decrement X from 7 to 6 and then load the accumulator with the value from address $1006.

Stepping through the table backwards, and keeping the count in the B register for convenience, our loop will now be:

```
        LDX     #64
        LDB     #64
LOOP    LDA     TABLE,-X
        DECB
        BGE     LOOP
```

The first of our two example programs shows a straightforward loop through a table of eight-bit values, in which we count the number of negative values. The count in accumulator B is also used as the offset from a fixed value in X. The second program shows both the index registers being used together, with a zero offset. It makes a copy of a character string from one location (possibly an input buffer) to another location where it will be stored. The string is of unknown length (although it will be less than 255 bytes) and will terminate with a Return character. When it is stored, the Return character will be deleted, and a byte indicating the string's length will be put at the beginning.

Negative Values Counter

- A table of eight-bit values is stored at location $3001. The number of values in the table is stored at $3000 and is assumed to be less than 256. This program counts the number of negative values in the table

```
TABLE   EQU     $3000   TABLE contains the number of values
        ORG     $1000
NEGS    FCB     0       NEGS is somewhere to keep the count of negative values
        LDB     TABLE
        LDX     #TABLE  The address of the table goes into X
LOOP    LDA     B,X     Get the last value from the table
        BGE     ENDLP   GOTO ENDLP if value is positive
        INC     NEGS    ELSE count it if the value is negative
ENDLP   DECB            Step down through table
        BGT     LOOP    Any more values? If so, GOTO LOOP
        SWI             ELSE return to operating system
        END
```

Character String Copier

- This program copies a string from the input buffer at $3000 to the next free string space, the address of which is given at $0010

```
CR      EQU     13      ASCII value of carriage return character
STRING1 EQU     $3000   Address of the input buffer
STRING2 EQU     $0010   STRING2 holds the address of the free space
        ORG     $1000
        LDX     #STRING1 Address of source string (i.e. $3000 at first) loaded into X
        LDY     STRING2 Address of destination string (i.e. address stored at $10,$11) into Y
        TFR     Y,U     Transfer the address of the length byte into U
        LDA     #0      Store zero at first byte of
        STA     ,Y+     destination string
LOOP    LDA     ,X+     Get next character from input buffer
        CMPA    #CR     Is it a Return character?
        BEQ     FINISH  Stop if it is
        STA     ,Y+     ELSE copy it (and)
        INC     ,U      Add one to the length
        BRA     LOOP    Next character
FINISH  SWI             Return to operating system
END
```